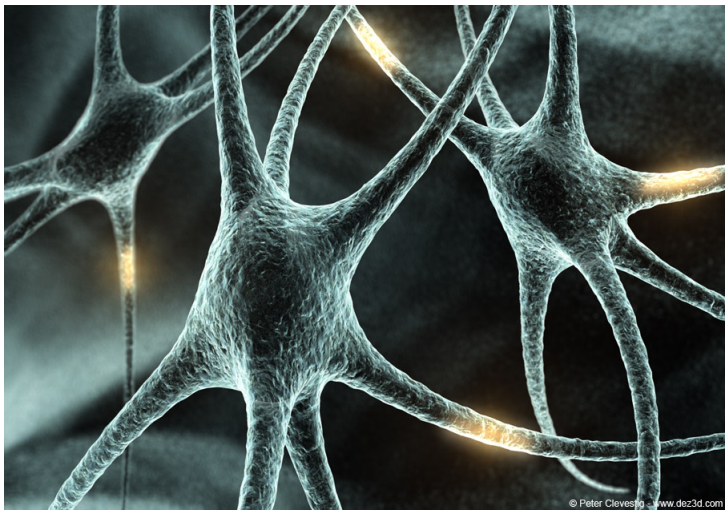
Shape Optimisation in Aeronautical Applications using Neural Networks



Alan Daring

MEng Thesis

Imperial College of Science, Technology and Medicine, London UK
Department of Aeronautics

6th June 2007

Abstract

An optimisation methodology based on neural networks was developed for use in 2D optimal shape design problems. Neural networks were used as a parameterisation scheme to represent the shape function, and an edge-based high-resolution scheme for the solution of the compressible Euler equations was used to model the flow around the shape. The global system incorporates neural networks and the Euler fluid solver into the C++ Flood optimisation framework containing a library of optimisation algorithms. The optimisation scheme was applied to a minimal drag problem in an unconstrained optimisation case and a constrained case in hypersonic flow using evolutionary training algorithms. The results indicate that the minimum drag problem is solved to a high degree of accuracy but at high computational cost. For more complex shapes, parallel computing methods are required to reduce computational time.

Acknowledgements

I would like to thank Roberto Flores for his help with the PUMI finite element solver and a special thanks to Roberto Lopez for his continual patience, support and guidance in the overall project as well as help with the neural network implementation. I would also like to express my gratitude to Dr. Cotter from Imperial College, and Prof. Bugeda and Prof. Oñate from CIMNE for making the project possible and their supervision throughout.

Contents

List of Figures

Chapter 1

Introduction

Optimisation techniques are used in engineering design processes to enhance the performance and properties of a product or component. This can be crucial in maintaining competitiveness in world markets. The use of optimisation methods in aeronautical applications has clear benefits and can be applied to much of the component design process. Often the aim is to find the most optimal component shape that is able to deliver the desired level of performance, subject to various constraints, e.g., total weight, size or cost, which must be satisfied. However, as aircraft grow more advanced, components must meet multiple objectives whilst satisfying more constraints. This need for multi-disciplinary approaches to engineering design has lead to increasingly complex models, for which flexible, automated environments are required. There is therefore a growing need for flexible problem solving environments, that can work as 'black-box' optimisers to integrate all the tools necessary in the optimisation process.

1.1 Background

The development of an advanced optimisation framework has been proposed by AIRBUS Spain (project DRAGON) [1]. The overall objective of this project is to develop and implement a global parametrisation scheme, an innovative optimisation software library, and advanced algorithms and techniques for parallel objective function evaluation, and to integrate them into an existing environment. The performance of the new system will be demonstrated for transonic drag reduction using unstructured finite element Euler and Navier-Stokes codes. Optimal shapes for minimum drag will be obtained for one or multiple objectives. The work will concentrate on the application of the methodology developed to the aerodynamic design of aircraft wings and more complex systems proposed by AIRBUS.

1.2 Objectives

This project is a preliminary study for the DRAGON project. It concentrates on the development and application of the methodology to simple 2D cases to provide an initial evaluation of the global system. The objective of the project is to integrate and to demonstrate the possibilities of an aerodynamic shape optimisation framework that combines neural network theory, a two-dimensional Euler code solver and a library containing optimisation algorithms. The neural network is used to represent the 2D shape function, which is manipulated by the optimisation algorithms based on the analyses performed by the finite element fluid solver. Along with integrating the system components, steps are taken to compare and certify the algorithms' behaviour, and to tune the algorithm parameters to the problem at hand. The Flood library developed at CIMNE containing the database of optimisation algorithms, implements the multilayer perceptron in the C++ programming language [2]. This will be used as the hosting environment for developing a comprehensive class library which will then be used to solve an unconstrained and constrained shape optimisation problem.

1.3 Neural Networks and Optimal Shape Design

Due to their variational nature, optimal shape design problems are difficult to solve and the only practical technique is to approximate the solution using direct methods. In [3], an optimisation procedure was carried out using Bézier curves defined by several control points to represent an aerofoil shape. Here, it was shown that Bézier curves are well suited to representing smooth curves but have weaker approximation properties as the order of the curve is increased. A similar study by Fung et al. involved varying three design parameters in standard polynomial equations that characterise NACA aerofoils, to validate evolutionary optimisation algorithms [4].

Conventionally, neural networks are used as non-linear data modeling tools to find solutions to function regression problems or pattern recognition problems. These are inverse problems where the multilayer perceptron is required to minimise the error function of the free parameters to obtain a given target. In [5], however, the learning problem for the multilayer perceptron is formulated in terms of objective functional rather than a concept of error function, which provides a direct method for the solution of general variational problems such as shape optimisation. Neural networks were shown to have much better approximation properties than polynomials or Bézier curves, and can be used to represent complex shape functions with a large number of parameters. In an aeronautical application, this could be extremely beneficial in designing completely new concepts as well as in modifying existing components.

Chapter 2

Neural Networks Theory

A neural network is a computing paradigm that is based on cortical structures of the brain. It consists of interconnected processing elements (neurons) that work together to produce an output function. Each neuron uses a set of inputs received from other neurons to compute the output and the neurons act in parallel with each other. The behavior of the network is determined by the individual neurons, the network connectivity, the weights associated with the interconnections between neurons, and the activation function of each neuron. Neural networks are traditionally used as non-linear data modeling tools to find solutions to function regression problems or pattern recognition problems. However, in this work we will use neural networks to approximate the shape function in the design space.

2.1 The Perceptron Neuron Model

A neuron is the basic information processing unit in a neural network [6]. The perceptron is the characteristic neuron model in the multilayer perceptron. Figure 2.1 shows a graphical representation of the perceptron which transforms the n inputs into a single output signal y.

The perceptron is made up from three basic elements:

1. A set of free parameters $\underline{\alpha} = (b, \mathbf{w})$
2. The combination function h
3. The activation function g

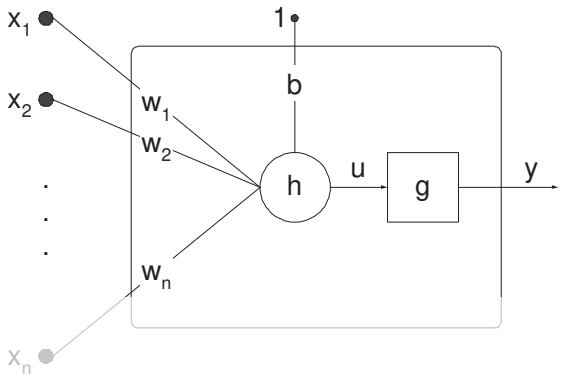


Figure 2.1: Perceptron Neuron model: The net input signal u is computed as a function h of the input signals $\mathbf{x}$ and the free parameters $(b, \mathbf{w})$. The input signals are received from neighbouring neurons and each input has a weight, or gain, w_i associated with it. Lastly, an activation function g produces the output signal y from the net input signal.

The combination function h combines the input signals and free parameters to produce a single net output signal u. It calculates the product of the input vector $\mathbf{x} = (x_1, \ldots, x_n)$ and the synaptic weight vector $\mathbf{w} = (w_1, \ldots, w_n)$ to find u. The weight can be positive or negative depending on the excitatory or inhibitory nature of the particular neuron. A bias b is then added to this output signal and can be represented as a synaptic weight connected to an input fixed to $+1$. The output signal of the neuron y is then determined as a function of g of the net input u. Two of the most popular activation functions are the sigmoid function, $g(u) = tanh(u)$, and the linear function, $g(u) = u$ [8].

The perceptron neuron model is extremely useful for accurately approximating functions. It can be viewed, mathematically, as a parameterised function space V from an input $X \subset \mathbf{R}^n$ to an output $Y \subset \mathbf{R}$ [5] (where $\mathbf{R}$ is a set of real numbers). The free parameters of the neuron $(b, \mathbf{w})$ determine a particular function within the function space V. Thus, the dimension of V is $n + 1$ (where n is the number of inputs). The elements of the function space which a perceptron can define are of the form,

$$\begin{aligned} y : \mathbf{R}^n &\rightarrow \mathbf{R} \\ \mathbf{x} &\mapsto y(\mathbf{x}; b, \mathbf{w}), \end{aligned}$$

where

$$y(\mathbf{x}; b, \mathbf{w}) = g\left(b + \sum_{i=1}^{n} w_i x_i\right). \tag{2.1}$$

Single neurons are able to perform simple tasks, but the power of the neuron comes from its collective behavior in a network where all neurons are interconnected.

2.2 The Multilayer Perceptron Model

Although a single neuron can perform certain simple tasks, the power of neural computing comes from connecting many neurons in a network architecture. Network architectures can vary in terms of the number of neurons, the arrangement of the neurons, and the connectivity between them [9].

The multilayer perceptron has a network space architecture known as a feed-forward architecture. This characteristically consists of several layers of neurons:

1. A set of sensorial nodes (which constitute the input layer)
2. One or more hidden layers of neurons
3. A layer of neurons (which constitute the output layer)

Each neuron in one layer is directly connected to the neurons in the next layer. Information is 'fed-forward' from one layer to the next, i.e. from the input layer via the hidden layers to the output layer. Figure 2.2 shows a multilayer perceptron with n inputs, one hidden layer with h_1 neurons and m neurons in the output layer.

The multilayer perceptron, by the same token as with a single perceptron, covers a parameterized function space V from an input $X \subset \mathbf{R}^n$ to an output $Y \subset \mathbf{R}^m$ [5].

Again the free parameters in the network define a particular function, or element of V. The free parameters can be represented by a free parameter vector with s dimensions, $\underline{\alpha} = (\alpha_1, \ldots, \alpha_s)$. The elements of the function space which the multilayer perceptron in Figure 2 can define are of the form:

$$\begin{aligned} \mathbf{y}: \quad \mathbf{R}^n &\rightarrow \mathbf{R}^m \\ \mathbf{x} &\mapsto \mathbf{y}(\mathbf{x}; \underline{\alpha}), \end{aligned}$$

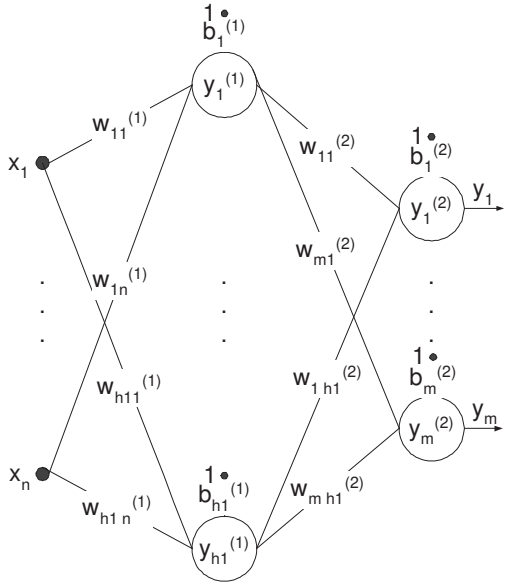


Figure 2.2: A two-layer perceptron with n inputs, one hidden layer with h_1 neurons and m neurons in the output layer.

where

$$y_k(\mathbf{x};\underline{\alpha}) = g^{(2)}\left(b_k^{(2)} + \sum_{j=1}^{h_1} w_{kj}^{(2)} \cdot g^{(1)}\left(b_j^{(1)} + \sum_{i=1}^{n} w_{ji}^{(1)} x_i\right)\right), \tag{2.2}$$

for $k = 1, \ldots, m$. It is also possible to modify this function to include lower and upper bounds, boundary conditions, or independent parameters, for example.

Multilayer perceptron neural networks have universal approximation properties in that they can approximate any continuous function spanning from one finite dimensional space to another. This is possible to an arbitrary level of accuracy by a multilayer perceptron with just one hidden layer of sigmoid neurons and an output layer of linear neurons (providing there are a sufficient number of hidden neurons available).

2.3 The Objective Functional

Conventionally, the goal of design is to find an acceptable value for some functional, whilst satisfying any other requirements or constraints associated with the problem.

There is often more than one adequate solution, and the aim of optimisation is to find the best of these solutions. The criterion used to compare different solutions (and enable optimisation) is known as the objective functional. The optimal solution is one that minimises (or maximises, if that is the goal) the objective functional of the function space that the network spans. In other words, the objective functional defines the task for the neural network and provides a reference for comparison which enables the multilayer perceptron to learn. An objective functional for the multilayer perceptron is of the form

$$\begin{aligned} F: \quad V &\rightarrow \mathbf{R} \\ \mathbf{y}(\mathbf{x};\underline{\alpha}) &\rightarrow F[\mathbf{y}(\mathbf{x};\underline{\alpha})] \end{aligned}$$

2.3.1 Unconstrained Variational Problems

Variational Problems for the multilayer perceptron in which the solution, $\mathbf{y}^*(\mathbf{x};\underline{\alpha}^*)$, has no constraints imposed on it, can be formulated as follows [6]:

If all the functions, $\mathbf{y}(\mathbf{x};\underline{\alpha})$, that the multilayer perceptron can define are within the s-dimensional function space, V, then the aim here is to find a vector of free parameters,

$$\underline{\alpha}^* \in \mathbf{R}^s$$

that parameterize a function,

$$\mathbf{y}^*(\mathbf{x};\underline{\alpha}^*) \in V$$

for which the functional defined on V,

$$F[\mathbf{y}(\mathbf{x};\underline{\alpha})] \tag{2.3}$$

is at a minimum or maximum value.

2.3.2 Constrained Variational Problems

In more complicated variational problems for the multilayer perceptron, the solution, $\mathbf{y}^*(\mathbf{x};\underline{\alpha}^*)$, must satisfy a set of constraints, $C_i[\mathbf{y}^*(\mathbf{x};\underline{\alpha}^*)] = 0$, for $i = 1, ..., l$ (where l is the number of constraints). However, this is not possible to find due to the inexact nature of numerical methods. Instead, each constraint is represented in the objective functional by a penalty term, reducing the constrained problem into an unconstrained problem. If a solution does not satisfy a constraint to an acceptable degree of accuracy,

then the penalty term adds a large positive or negative value to the objective functional, so that the solution is recognized as infeasible. This problem can be formulated as follows [6]:

Again assuming that all the functions $\mathbf{y}(\mathbf{x};\underline{\alpha})$ *that the multilayer perceptron can define are within the* s*-dimensional function space,* V*, then now the aim is to find a vector of free parameters,*

$$\underline{\alpha}^* \in \mathbf{R}^s$$

that define a function,

$$\mathbf{y}^*(\mathbf{x};\underline{\alpha}^*) \in V$$

for which the functional defined on V*,*

$$F[\mathbf{y}(\mathbf{x};\underline{\alpha})] + \sum \rho_i \|C_i[\mathbf{y}(\mathbf{x};\underline{\alpha})]\|^2$$

for $i = 1, \ldots, l$ *and* $\rho_i > 0$*, is at a minimum value.*

In this way, the variational problem is reduced to a function optimisation problem. Note that in order to maximize the objective functional, the sign is simply changed so that the problem is converted into a minimization of $-F[\mathbf{y}(\mathbf{x};\underline{\alpha})] + \sum \rho_i \|C_i[\mathbf{y}(\mathbf{x};\underline{\alpha})]\|^2)$.

The penalty term ratios, ρ_i, are parameters used to control the penalty terms. If ρ_i is large then the solution is found when $C_i[\mathbf{y}(\mathbf{x};\underline{\alpha})]$ is small. When the solution lies well into the infeasible region this term is large, making the objective functional large and therefore the solution is rejected. When $C_i[\mathbf{y}(\mathbf{x};\underline{\alpha})]$ is sufficiently small, there is a negligible value added by the penalty term and $F[\mathbf{y}(\mathbf{x};\underline{\alpha})]$ can then be minimized. Therefore as $\rho_i \to \infty$, the solution to the reduced constrained problem will approach the solution to the unconstrained problem given by Equation 2.3. The penalty term ratios must be carefully chosen so that the constraints are met to a sufficient level of accuracy and so the solution is found in an acceptable length of time.

2.4 The Training Algorithm

The training (or learning) algorithm is the method used by the multilayer perceptron to find the solution to the optimisation problem. It involves continually adjusting the free parameters in the network by small increments looking for a parameter vector within the s-dimensional function space, V, that optimises the objective function. The free

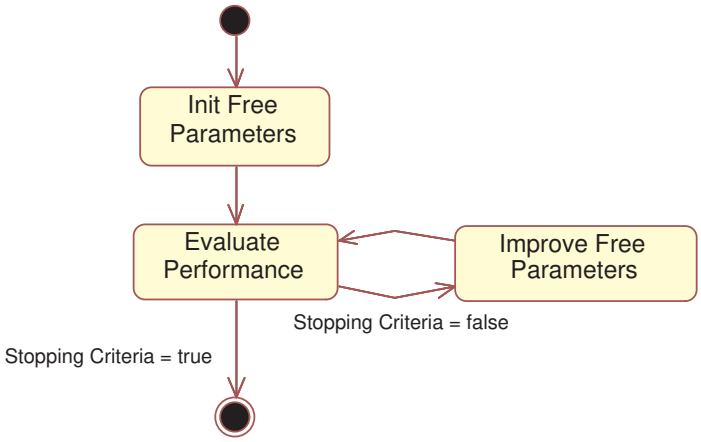


Figure 2.3: State diagram of the learning process in the multilayer perceptron

parameters can be randomly initialised and at each epoch (iteration step) they are changed so that the objective function becomes smaller each time. That is,

$$\alpha^{(\tau+1)} = \alpha^{(\tau)} + \Delta\alpha^{(\tau)}$$

so that,

$$f(\alpha^{(\tau+1)}) < f(\alpha^{(\tau)})$$

where the epoch number is denoted by τ.

This can be summarized as in Figure 2.3. The stopping criterion could be a maximum number of epochs or computational time, or when the objective function ceases to decrease by more than a specified value after each epoch (i.e. the optimum value has been reached).

There are numerous algorithms available for training neural network models, each with different advantages and disadvantages associated with them. The two used for the purposes of this project were the *quasi-Newton method* and the *evolutionary algorithm*.

2.4.1 The Quasi-Newton Training Algorithm

The quasi-Newton method is a first-order order training algorithm which means it uses both the objective function and its gradient vector to find the optimum solution to a problem. Quasi-Newton methods are based on Newton's method except they do not need to find the second derivatives (known as the Hessian matrix) of the objective function, but instead approximate them using successive gradient vectors to reduce the amount of computation per iteration [11].

Gradient-based training algorithms take the derivative of the objective function with respect to the free parameters in the network and then adjust those parameters such that the objective function is reduced (thus going downhill on the surface of the function where the gradient is less steep). Therefore the method assumes that the function can be locally approximated as a quadratic in the region around the optimum. However, there is no guarantee that this value is the global optimum (the global minimum is the smallest value over the entire function domain where as the local minimum of a function is the smallest value within a given region). The advantages of gradient methods such as this one, however, are that if they converge, they usually converge faster than non-gradient methods [2].

When the desired output of the multilayer perceptron is known the objective function gradient is usually found using backpropagation [2]. The network is trained by minimising the error between the actual output values and a target output values corresponding to the given input values. The weights of each connection are then adjusted in order to reduce the value of the error. In other words, backpropagation is used to calculate the gradient of the error of the network with respect to the free parameters. However when the target output is unknown as is the case in this project, direct methods are required so the gradient of the objective function is determined numerically. The derivative of the objective function f at a point α_i can be approximated after an incremental change in each parameter in turn by using the central difference approximation,

$$\frac{\partial f}{\partial \alpha_i} = \frac{f(\alpha_i + h) - f(\alpha_i - h)}{2h} + \mathcal{O}(h^2), \tag{2.4}$$

for $i = 1, 2, \ldots, s$, when the step size h is small. It can be seen that the truncation error of the central difference approximation is order of $\mathcal{O}(h^2)$, hence the approximation is sufficiently accurate when h is small.

2.4.2 The Evolutionary Training Algorithm

Evolutionary algorithms are zero-order training algorithms which use the objective function only - gradient information of the objective function is not required. It is a stochastic method which incorporates randomness, and is based on the biological evolution demonstrated in nature (quasi-Newton is a deterministic algorithm which will produce the same results if initiated at the same point of the search space). Evolutionary algorithms eval-

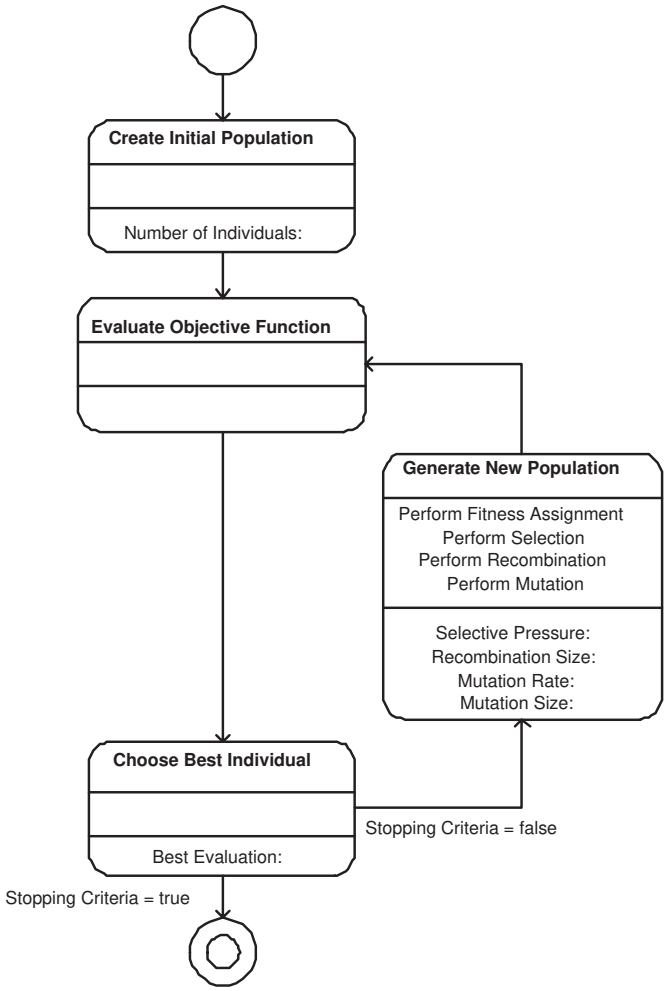


Figure 2.4: Solution Process in Evolutionary Algorithm Methods

uate a set of approximations to a solution (known as a population of individuals) and

use the best ones to produce new, better approximations. They model natural processes, such as selection, recombination, and mutation to carry out this optimisation task. The solution process carried out in this method is shown in Figure 2.4. At the beginning of the computation the first generation population is randomly initialized and the objective function is evaluated for each individual. Each individual is then assessed and *assigned a fitness* value based on their suitability to the problem. The *selection* operator then determines which individuals will be used to create the new improved population based on these fitness values. The *recombination* process models mating in the natural world, and combines information from the parent population of chosen individuals to form the new generation of offspring. In addition, parameters, or genes, can undergo some degree of mutation to incorporate new information into the offspring generation. So by applying the survival of the fittest concept what results is an evolutionary progression, with each generation of individuals becoming closer to the optimum solution until the optimisation criteria are met.

Evolutionary algorithms have several advantages compared to gradient based optimisation methods like the quasi-Newton method. Gradient methods may not converge to a solution unless all parameters are carefully chosen, and calculating the gradient of the objective function can be time consuming and unreliable in the presence of noise. Evolutionary algorithms are extremely robust and much more straightforward to apply as they need minimal guidance and only rely on the objective function. The evaluation of several generations of many individuals means that a large search space can be explored and that the global optimum will always be found if it exists (given enough individuals and generations). This use of populations of potential solutions that characterize the method, also allows for the evaluation of individuals to be performed in a parallel manner to reduce computational time.

The main drawback associated with this method, however, is the need for a large number of calculations. Each individual is analysed to evaluate the objective function which comes at the expense of high computational cost. Furthermore, although a very robust method, poorly chosen operators and parameters can still be problematic and inhibit convergence. Having said this, providing the parameters are correctly chosen, the choice of operator schemes for selection, recombination and mutation becomes a matter of preference as all methods will converge to the global solution. For a detailed explanation on evolutionary algorithms and all operators see [12]. For this project the following operators were chosen:

Fitness Assignment Method: *Linear Ranking*

Linear ranking is a simple and reliable method to assign fitness values to individuals in a population. The population is organised in order of a rank which is determined by the objective function evaluation. The probability of one individual being chosen for recombination is linearly proportional to their rank in the population. The fitness value for an individual is calculated as follows [12]:

Let N represent the number of individuals in the population (i.e. population size), R the rank or position of an individual in this population (with the least fit individual with $R = 1$, and the fittest individual with $R = N$) and let the selective pressure be denoted by SP.

$$Fitness(R) = 2 - SP + 2(SP - 1)\frac{(R-1)}{(N-1)} \tag{2.5}$$

where the selective pressure lies in the interval $[1.0, 2.0]$. Using a ranking system allows for a uniform fitness scale across the population and is more robust than proportional fitness assignment where the probability of any individual being chosen for recombination is proportional to the objective function evaluation. The latter method can suffer from stagnation (no convergence) or premature convergence (converges too fast) when the fitness values are extreme.

In this work, $SP = 1.5$.

Selection Method: *Stochastic Universal Sampling*

In stochastic universal sampling each individual is assigned a segment of a continuous line of unit length 1. The length of the segment is proportional to the fitness value of the particular individual. A set of equally spaced markers are then spread out along the whole line to select the individuals to be recombined. The individual is chosen if a marker lies within the segment corresponding to that individual. Therefore if M is the number of individuals to be selected, then there are M markers and the spacing between each marker is $1/M$. The position of the first marker is determined randomly within the spacing $1/M$. In this way is the fittest individual occupies the largest interval and has a higher probability of being selected than the least fit individual whose segment spans the shortest distance on the line [12].

In this work, $M = N/2$.

Recombination Method: *Intermediate*

In intermediate recombination the free parameters of the offspring are chosen somewhere around and between the free parameter values of the parents. The size of the area for possible offspring is determined by the recombination size A. When $A = 0$ the area spanned by the offspring is theoretically the same size as the hypercube defined by the parents. In practice, this area is reduced with every generation due to the fact that the offspring are rarely generated on the border of the hypercube. A value of $A = 0.25$ has been shown to create a hypercube the same size as the parent's when taking this shrinkage into account. Offspring are produced according to the rule:

$$\alpha_i^O = \alpha_i^P k + \alpha_i^P (1 - k) \tag{2.6}$$

for $i = 1, 2, \ldots, s$ where O denotes offspring, P denotes parent, and where k is a scaling factor chosen uniformly at random over an interval $[-A, 1 + A]$ for each i [12].

Mutation Method: *Normal*

Every offspring is subject to a mutation to introduce new information into the mating pool just like in natural evolution. It is used as a way to generate individuals outside of the range of the existing population to explore new regions of the search space and help the algorithm escape local minima. In normal mutation, a random number with normal distribution is added or subtracted from free parameters. The *mutation range* determines the range of values in the distribution. The further outside the range, the lower the probability is for a parameter to be generated there. The probability of a free parameter undergoing mutation is known as the *mutation rate* and is inversely proportional to the number of free parameters [12]. Here, the mutation rate is set so that approximately one parameter per individual is mutated.

In this work, *Mutation Rate* $= 1/$*No.of Parameters Mutation Range* $= 0.1$

Chapter 3

Fluid Solver Theory

In order to model the flow over the shape and evaluate the objective functional, an Euler solver developed at CIMNE, named PUMI, is used. This is an edge-based high resolution scheme used to solve compressible Euler equations on an unstructured finite element grid [13].

The general scheme involved in solving the Euler equations is characterised by the following features [15]:

- An adequate discretisation of the solution space to ensure accuracy and convergence
- A high-order interpolation scheme to determine the variables at the midpoint between nodes.
- A Riemann Solver to stabalise the solution in the presence of discontinuities.
- a high-order time-integration method.

The basic theory behind the constituents applied in this work is presented in this section.

3.1 The Euler equations

The Euler equations express the conservation of mass, momentum and energy in a compressible, inviscid and non-conducting fluid.

That is, the rate of change of the conservative variable in a fixed control volume, Ω, plus the net flux of flow out of the volume, F, is equal to zero.

$$\frac{\partial}{\partial t}\int_{\Omega}\phi d\Omega = -\int_{\Gamma}\phi\, V\cdot\hat{n}d\Gamma$$

where $\hat{\mathbf{n}}$ is the outer pointing normal vector to the boundary of Ω. In 2D, and neglecting source terms, the general Euler equations can be reduced, in conservation vector form, to [16],

$$\frac{\partial U}{\partial t}+\frac{\partial F_1}{\partial x_1}+\frac{\partial F_2}{\partial x_2}=0 \tag{3.1}$$

where U is the vector of states and F_i is the inviscid flux vector and are defined by,

$$U=\begin{bmatrix}\rho\\ \rho u_1\\ \rho u_2\\ \rho E_t\end{bmatrix}\qquad \mathbf{F_1}=\begin{bmatrix}\rho u_1\\ \rho u_1^2+p\\ \rho u_1u_2+p\\ (\rho E_t+p)u\end{bmatrix}\qquad \mathbf{F_2}=\begin{bmatrix}\rho u_x\\ \rho u_1u_2+p\\ \rho u_2^2+p\\ (\rho E_t+p)u\end{bmatrix}$$

where E_t is the total specific energy of the fluid consisting of the specific internal energy, and the kinetic energy,

$$E_t = C_vT+\tfrac{1}{2}u^2$$

The equation of state is required to close the equation system and relate the specific internal energy $e = C_vT$ to p and ρ

$$p=R\rho T=\tfrac{C_p-C_v}{C_v}\rho e=\rho(\gamma-1)\left[E_t-\tfrac{1}{2}u^2\right]$$

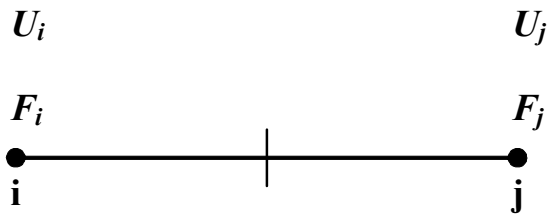
where $\gamma = C_p/C_v$ is equal to the ratio of specific heats and is taken as $\gamma = 1.4$ in this application. Initial and boundary conditions are assigned to solve the system of equations over a closed domain with boundary for a given time. Considering the fluid in a domain, Ω, with boundary, $\Gamma = \Gamma_0 \cup \Gamma_n$

$$\begin{aligned} U(x,0)&=U_0(x) && t=0, x\in\Omega\\ U(x,t)&=U_n(x) && \geq 0, x\in\Gamma_0\\ F^n=F\cdot\hat{n}&=\bar{F}^n && t\geq 0, x\in\Gamma_n\end{aligned}$$

3.2 Roe's Approximate Riemann Solver

The flow solver used in the PUMI application is Roe's approximate Riemman solver [13]. As the fluid is modeled as compressible, shocks can occur where there is an abrupt change in the fluid variables. The idea behind this method is to solve what is known as the Riemman problem, where a discontinuity in the state variables occurs at the interface of two piecewise constant states U_L and U_R. The Riemman solver approximates the non-linear interaction between U_L and U_R to reduce the computational cost with sufficiently accurate results. This numerical dissipation ensures stability in the solutions and avoids the formation of spurious numerical oscillations.

Figure 3.2 shows an edge between two nodes i and j with flux and states associated with them. We need to approximate the flux between the nodes.



The first order flux is defined for each edge by,

$$\tilde{F_{ij}} = \frac{F_j + F_i}{2} - |A(U_i, U_j)|\frac{(U_j - U_i)}{2} \tag{3.2}$$

where Ui and U_j is the vector of conservative variables at the edge nodes i and j and $|A|$ is the absolute value of the Roe matrix used to approximate the flux between the two states [13]. The quasi-linear form of the Euler equations is therefore given by,

$$\frac{\partial U}{\partial t} + A_1\frac{\partial U}{\partial x_1} + A_2\frac{\partial U}{\partial x_2} \tag{3.3}$$

Evaluating A can be very complicated, particularly in 2D and 3D. PUMI uses the following concept to reduce computational time.

As the Euler equations are hyperbolic, exists a diagonal matrix Λ_{ij} containing real eigenvalues of the Jacobian matrix, A, and an associated matrix, R_{ij}, made up from linearly independent eigenvectors. Therefore the Roe matrix for the edge e_{ij} can be expressed as,

$$A = R_{ij}^{-1}\Lambda_{ij}R_{ij}$$

where $\Lambda_{ij} = diag(\lambda_1, \lambda_1, \lambda_2, \lambda_3)$ With the definition,

$$\tilde{U} = R^{-1}U$$

Equation 3.3 gets the characteristic form,

$$\frac{\partial \tilde{U}}{\partial t} + \Lambda \frac{\partial \tilde{U}}{\partial x}$$

This is now a set of independent equations,

$$\frac{\partial \tilde{U_k}}{\partial t} + \lambda_k \frac{\partial \tilde{U_k}}{\partial x}$$

lll where $k = 1 \dots m$ and m is the number of eigenvalues. Three distinct values for the wave propagation speed are then obtained,

$$\lambda_1 = \lambda_1 = \hat{u}_{ij}$$
$$\lambda_2 = \hat{u}_{ij} + \hat{c}_{ij}$$
$$\lambda_3 = \hat{u}_{ij} - \hat{c}_{ij}$$

where $\hat{u}_{ij}$ is the velocity along the edge and $\hat{c}_{ij}$ is the average speed of sound.

General expressions for the Jacobian, eigenvalues and eigenvectors matrices can be found in [5].

Although this scheme has good properties of monotonicity, it suffers from severe inaccuracies because of the high level of truncation error, so a higher order scheme is needed for when the flow does not contain discontinuities.

3.3 High-Resolution Scheme

In order to obtain high-order approximations between the nodes, a piecewise linear reconstruction of the variables at neighbouring nodes is applied. To do this, the MUSCL scheme was implemented.

Figure 3.1 represents the high order approximation. The gradient, ∇U_i at point i is calculated using a central difference scheme,

$$\nabla U_i = \frac{U_j - U_{i-1}}{2l_{ij}} \qquad \Rightarrow \qquad 2l_{ij}\nabla U_i = U_j - U_{i-1} + U_i - U_i$$
$$\Rightarrow \qquad 2l_{ij}\nabla U_i = \Delta_i^- + \Delta_{ij}$$

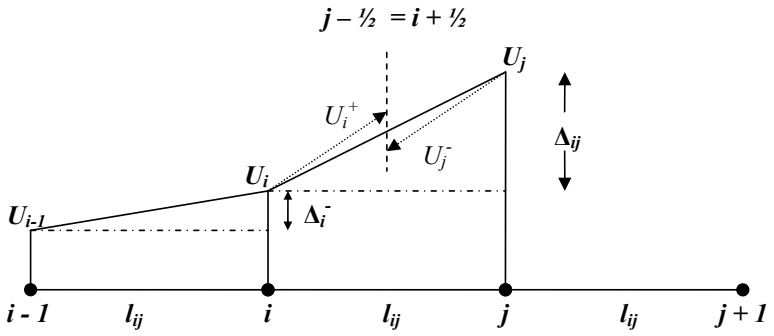


Figure 3.1: High-order approximation

Therefore the difference operator at node i can be expressed as,

$$\Delta_i^- = 2l_{ij}\nabla U_i - \Delta_{ij}$$

A similar process is used at node j to obtain,

$$\Delta_j^+ = U_{j+1} - U_j = 2l_{ij}\nabla U_j - \Delta_{ij}$$

The MUSCL scheme then approximates the interface value between all nodes using the following,

$$U_i^+ = U_i + \frac{1}{4}[(1-k)\Delta_i^- + (1-k)\Delta_{ij}] \tag{3.4}$$

$$U_j^- = U_j - \frac{1}{4}[(1-k)\Delta_j^- + (1-k)\Delta_{ij}] \tag{3.5}$$

where k is used to determine the order of approximation. Here $k = -1$ was used for a second order upwind scheme.

When $U_j^+ \simeq U_i^-$ i.e. the flow is smooth, the dissipative term

$$|A(U_i^-, U_j^+)|\frac{(U_j^+ - U_i^-)}{2} \simeq 0$$

and can be omitted from Equation 3.2 so that,

$$\tilde{F_{ij}} = \frac{F_j + F_i}{2} \tag{3.6}$$

When U_j^+ is very different to U_i^- non-linear instabilities and oscillations occur across the discontinuity, and Roe's approximate Riemman solver must be used as outlined in the previous section.

Flux limiters, s_i and s_j, based on the Van Albada model [18], are therefore included in Equation 3.7 to ensure an oscillation-free solution is obtained across discontinuities and higher order scheme is used in smooth flow for greater accuracy.

$$U_i^+ = U_i + \frac{s_i}{4}[(1 - ks_i)\Delta_i^- + (1 - ks_i)\Delta_{ij}]$$
$$U_j^- = U_j - \frac{s_j}{4}[(1 - ks_j)\Delta_j^- + (1 - ks_j)\Delta_{ij}]$$

where

$$s_i = max\left\{0, \frac{2\Delta_i^-\Delta_{ij} + \varepsilon}{(\Delta_i^-)^2 + (\Delta_{ij})^2 + \varepsilon}\right\}$$

$$s_j = max\left\{0, \frac{2\Delta_j^-\Delta_{ij} + \varepsilon}{(\Delta_j^-)^2 + (\Delta_{ij})^2 + \varepsilon}\right\}$$

The small constant, ε, is included to avoid divisions by zero and the minimum value of the limiters is set to 0 as negative limiters would disrupt the solution.

Now, when calculating U_i^+ for example,

$$\begin{array}{rccccccc} \textit{Smooth Flow} & \Rightarrow & \Delta_{ij} \simeq \Delta_i^- & \rightarrow & s_i \simeq 1 & \rightarrow & U_i^+ \simeq U_i^+ \\ \textit{Shock} & \Rightarrow & \Delta_{ij} \gg \Delta_i^- & \rightarrow & s_i \simeq 0 & \rightarrow & U_i^+ \simeq U_i \end{array}$$

Therefore, in the presence of a sharp gradient, the dissipative term must be evaluated using Roe's approximate Riemman solver, but when the flow is smooth, Equation 3.2 reduces to Equation3.6 giving a high order approximation.

The high-order approximations then replace the first order flux given in Equation 3.2 to give,

$$\tilde{F_{ij}} = \frac{F_j - F_i}{2} - |A(U_i^+, U_j^-)|\frac{(U_j^- - U_i^+)}{2} \tag{3.7}$$

3.4 Time Discretisation

An explicit multistage Runge-Kutta time-marching scheme is implemented. This acts as a compromise between the explicit forward Euler scheme, where the time step, Δt must be small for stability (resulting in a large number of calculations), and an implicit

backward Euler scheme which requires an iterative procedure (again resulting in high computational cost). The scheme used is split into 3 stages.

$$U^{n+\alpha_i} = U_x + \alpha_i \Delta t \frac{\partial U}{\partial t}|_n \tag{3.8}$$

with

$$\alpha_1 = \frac{1}{4}, \qquad \alpha_2 = \frac{1}{3}, \qquad \alpha_3 = \frac{1}{2} \qquad \alpha_4 = 1.0$$

Splitting up the space allows the time step, Δt, to be larger.

3.5 Drag Calculation

Once the fluid properties have been determined, the drag can be calculated from the pressure. The drag is defined as the component of the pressure distribution in the freestream direction. It is found by integrating the pressure over the surface of the shape function.

$$D = -\int p.n_x d\Gamma$$

For a small section of the surface,

$$dD = -pn_x d\Gamma$$

Figure 3.2 shows the pressure distribution over surface $d\Gamma$,

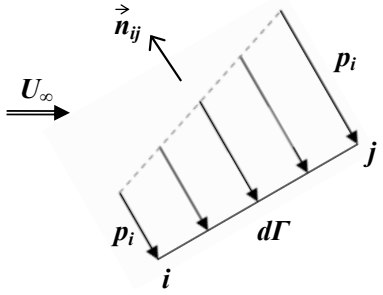


Figure 3.2: Pressure distribution between two nodes

Therefore the drag on cell edge ij on the surface of the shape is,

$$D_{ij} = -\frac{p_i + p_j}{2} \cdot d\Gamma_{ij} n_{x(ij)} \tag{3.9}$$

Chapter 4

Minimum Drag Problem

The minimum drag problem in this project is concerned with optimizing the 2D shape between two points to minimise the drag under specified flow conditions. This problem has obvious significance in the design of airfoils and other aeronautical applications. The aim is to apply the neural networks theory to train the shape function based on the continual analysis of the objective function using the fluid solver. This methodology will be used firstly in an unconstrained problem and following this, a constraint will be applied and the optimisation procedure will be repeated. Although a design problem found in practice is unlikely to be unconstrained, it is important to use unconstrained minimisation techniques before adding constraints to a design problem. These methods are much more robust and provide the basis for the constrained problems.

The application of the neural networks theory to the minimum drag problem has already been investigated through the design of an axisymmetric body of minimum drag with analytical solution [6], with encouraging results. In this study though, the objective functional (drag) was defined by an integral and was evaluated using Simpson's composite method. This numerical method uses several assumptions that reduce the accuracy of the approximation. By integrating the fluid solver into the optimisation process to evaluate the objective functional, the reliability and accuracy of the results will be increased. The performance of the edge-based finite element methodology has been tested on several academic simulations involving subsonic, transonic and supersonic flows [13]. All of them were solved with satisfactory accuracy.

4.1 Problem Definition

The pre and post processing system used in the analysis is GiD. GiD provides the tools for the definition of the geometry, condition assignment, mesh generation and visualization of the numerical results (See Appendix B). The Euler code, PUMI, presented in Chapter 3, is used for the flow analysis and is run within the GiD environment.

4.1.1 Geometry

The geometry consists of a semicircular control area. The aim is to find the shape joining the two base lines so that the drag is minimised. Figure 4.1 shows the basic geometry with a randomly generated shape function.

Initially, the shape was represented by 21 interconnected control points (as shown in the figure). After the methodology was validated, the number of control points was increased to 101 to obtain a smoother, more accurate shape. A fixed grid Eulerian approach is taken, with the shape defined by a function on a rectangular space. The starting shape function is defined by a randomly initialised neural network. The evolutionary algorithm will then search the entire function space to find the function which minimises the objective functional.

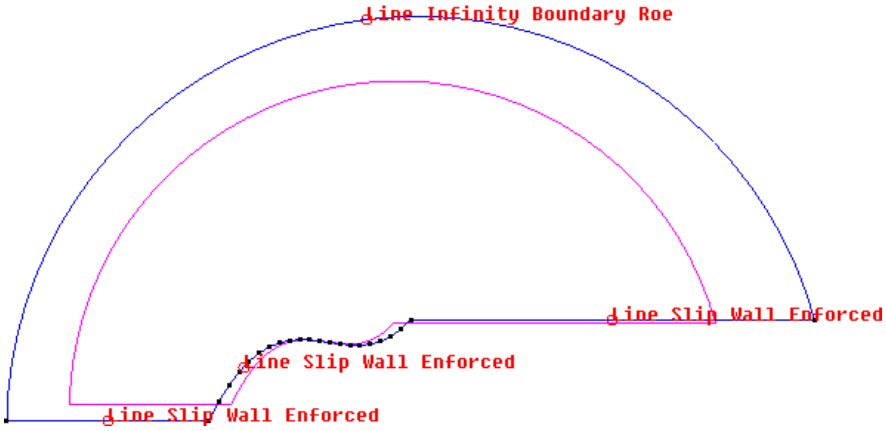


Figure 4.1: Problem Geometry and Conditions

No previous data is used when initialising the parameters. It is possible to use already well performing aerofoils to limit the search space, which would reduce the computational time significantly. However, this tends to result in very small modifications to known

standard designs. By searching a large function space there is more scope to discover completely new and novel designs which would otherwise not be considered.

In order to take decisions about the quality of each design it is necessary to compute the corresponding values of the objective function and the restrictions. This is carried out by performing a drag analysis for each individual, using a specific finite element mesh. In the analysis of each different design the computational cost and the quality of the solution are very much dependent on the quality of the finite element mesh used. One important ingredient of the numerical analysis is the strategy for the generation of a proper mesh for each design. Here, a three step approach was taken to make the optimisation process as efficient as possible. In the first step, an approximate solution is found using a relatively coarse mesh. This allows a quick and general search of the entire function space for the optimum solution. In the second step the mesh is refined to enable a more accurate evaluation of the objective function to be carried out. The extra number of calculations is balanced by the reduced search space. The third step involves a further improvement in the mesh to obtain a sufficiently accurate solution.

4.1.2 Conditions and Parameters

Modeled Flow

Infinity Boundary Roe condition is assigned to the boundary arc to the control area. This allows the flow to escape the control volume without the reflection of any flow properties at the boundary [14]. The baselines and the entire shape function are all assigned to be Line Slip Wall Enforced condition to neglect skin frictional forces. This means that the drag will only be measured on the shape function. In the flow model, the freestream Mach number is $M_\infty = 2$ therefore the flow is hypersonic. The full details on the control variables used by the Euler solver can be found in the Appendix.

Evolutionary algorithm

Table 4.1.2 provides a summary of the operators and parameters chosen for the evolutionary algorithm.

The operator parameters are chosen due to their statistically reliable performance [12]. The number of parameters determines the dimensions of the search space and is determined by the multilayer perceptron architecture. More neurons in the multilayer perceptron will have more parameters, which will be able to represent a greater number of designs which may not necessarily comply with standard designs. However, as previously discussed this comes at the cost of computational time. The choice of number

Training Operators	
Fitness Assignment Method	Linear ranking
Selection Method	Stochastic universal sampling
Recombination Method	Intermediate
Mutation Method	Normal
Training Parameters	
Population Size	60
No. of Free Parameters	10
Selective Pressure	1.5
Recombination Size	0.25
Mutation Rate	1/No. of Parameters
Mutation Range	0.1

Table 4.1: Evolutionary Algorithm Training Operators and Parameters

of individuals is a similar compromise. Each individual of the population describes one complete aerofoil shape and must be evaluated using the fluid solver. But if there are not enough individuals in a population, the algorithm may not explore enough of the solution space to find the global optimum.

4.1.3 Network Architecture

The network architecture used to represent the shape function is a multilayer perceptron with sigmoid hidden layer and linear output layer. The network must have one input and one output neuron. Three neurons are used in the hidden layer, i.e. a 1:3:1 multilayer perceptron layer is used as shown in Figure 4.2.

4.2 Procedure

An Application was created for both the unconstrained problem and the constrained problem in the Flood neural networks library, which acted as a central command environment for the entire optimisation procedure. The Flood library is a comprehensive class library which implements the multilayer perceptron in the C++ programming language [7]. Both applications were given access to the neural network classes, optimisation algorithms, GiD, and PUMI. Any or all of the neural network and optimisation parameters, can be modified from within these applications. A new object was also

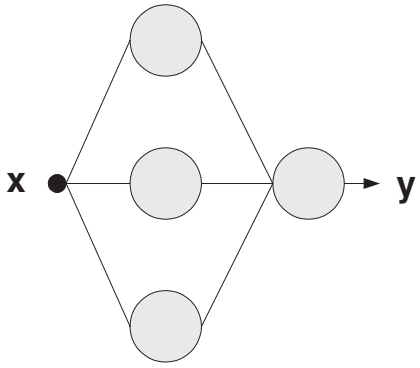


Figure 4.2: A neural network with 1 input neuron, 3 hidden neurons and 1 output neuron used to parameterize the shape function.

created in the Objective Functional class for both problem types. The basic procedure for both the unconstrained and constrained problems is identical. Figure 4.3 illustrates the optimisation process,

On running the Application within Flood, the multilayer perceptron object is called. This creates a multilayer perceptron object with one hidden layer of sigmoid neurons and an output layer of linear neurons. All the free parameters in the network are initialised with random values chosen from a normal distribution with a set mean and standard deviation. Initially, the entire search space is used. However, as the mesh is refined the free parameters can be initialised randomly, but close to, the predetermined shape obtained from the previous mesh to reduce the search space for a faster solution. This could also apply in an application where small modifications to a known design are required.

In order to take decisions about the quality of each design it is necessary to compute the corresponding values of the objective functional. This is carried out by performing a PUMI analysis for each individual, within the Objective Functional object. A batch file, *GiDBatchTemplate.bch*, written in GiD command language and attached in Appendix C, automatically generates the template geometry (excluding the shape function), mesh and all flow and boundary conditions. The template is rewritten into *GiDBatch.bch* where the individual shape function (determined by the current parameters) is inserted into a flagged section to complete the geometry. A file *batch.bat* then executes the Euler solver PUMI, *pumi*2*d*3.*exe*, which analyses the shape in the GiD environment and returns an evaluation of the objective functional. The output is saved in a post-processing file, *pumi*2*d.post.res*, so that the individual results can be represented visually.

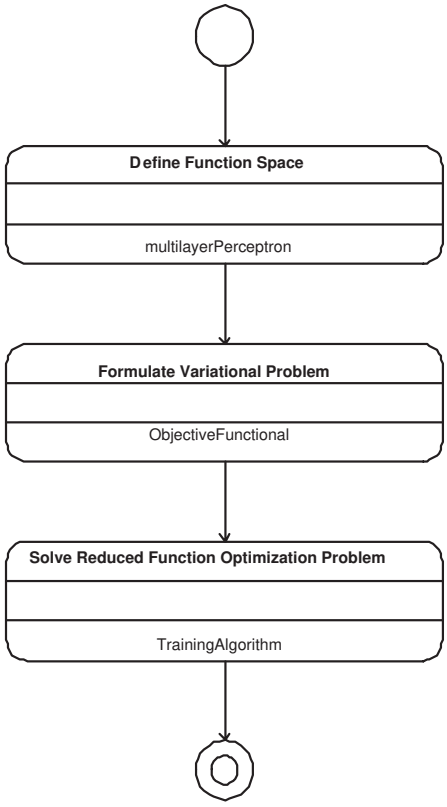


Figure 4.3: NeuralNetworkActivityDiagram

The solution is then trained by calling the Evolutionary Algorithm object or Quasi-Newton object from the Training Algorithm class. The stopping criterion for the Evolutionary Algorithm is the number of generations and for Quasi-Newton, the number of epochs.

The coordinates of the shape function are written into a *Results.dat* file with the vector of parameters provided in $MultilayerPerceptron.dat$. The history of the optimisation procedure (results after each evolutionary algorithm generation or quasi-Newton epoch) can also be viewed in the $EvolutionaryAlgorithmTrainingHistory.dat$ and $TrainingHistory.dat$ files.

4.3 Results

4.3.1 Unconstrained Problem

In the unconstrained problem the objective functional, which is to be minimised, is the drag,

$$F[\mathbf{y}(\mathbf{x};\underline{\alpha})] = D \tag{4.1}$$

Mesh 1 - Coarse Mesh

Initially a coarse mesh was used to obtain a fast, approximate solution in order to reduce the size of the search space quickly. An unstructured 5264 triangular element mesh is employed in the numerical simulation with a finer mesh on the boundary of the shape function so that the objective functional is much more sensitive to small changes in the shape. An evolutionary algorithm was used to train the multilayer perceptron for 10 generations. Quasi Newton was tested but did not always converge. Figure 4.4 shows the initial mesh implemented in the first run of the optimisation process.

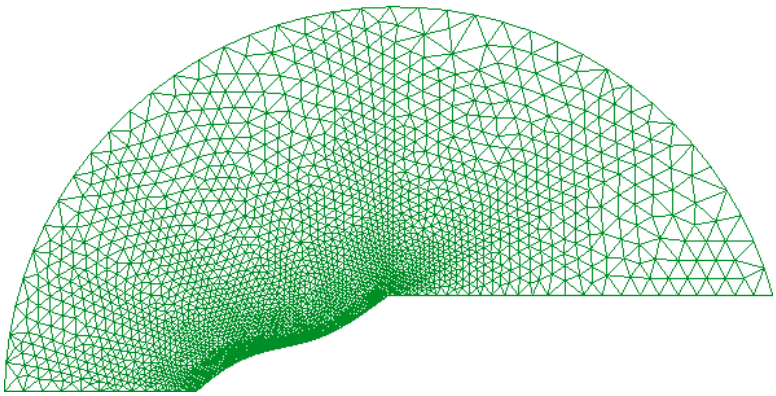


Figure 4.4: Initial Mesh

Mesh 2 - Intermediate Mesh

Initial results showed a region of high pressure at the 'leading edge' of the shape function which contributed significantly to the drag (see Figure A.7, Appendix A). The mesh was

therefore modified further so that it was more focused here where discontinuities in the solution field are expected to improve the accuracy of the results. Figure 4.5 shows the new mesh with 10118 triangular elements. The multilayer perceptron from the previous run was loaded so the algorithm searches around the solution found previously (parameters are initialised within a small distance from the previous solution)to ensure an efficient search of the function space is carried out. The evolutionary algorithm is now applied using 50 generations. Each generation consists of 60 individuals therefore the total number of objective functional evaluations is 3000. With the refined mesh the flow solver also has more calculations to carry out and due to the large number of evaluations, the run required around 120 hours of time in a Pentium D 3.00GHz CPU with 1.00GB RAM.

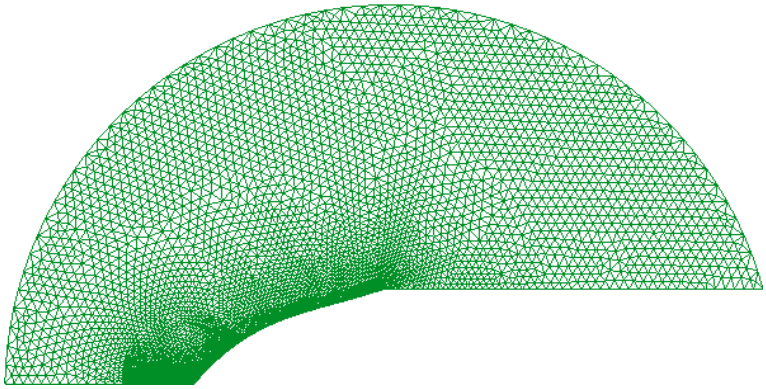


Figure 4.5: Intermediate Mesh

Mesh 3 - Refined Mesh

The mesh was made refined further around the shape function using a second surface so that the analysis would be more accurate in the region of discontinuity. This allows for a much more accurate evaluation of the objective function. The final unstructured mesh is shown in Figure 4.6 and uses 16620 triangular elements. At first the faster quasi-Newton algorithm was used to train the multilayer perceptron to focus on the solution. However, the solution was a minimal modification of the shape function found from the second run. The process was repeated using an evolutionary algorithm using 30 generations. There was a much greater improvement in the solution. This may have been due to noise around the minimum which creates many local minima that could 'trap' the quasi-Newton method.

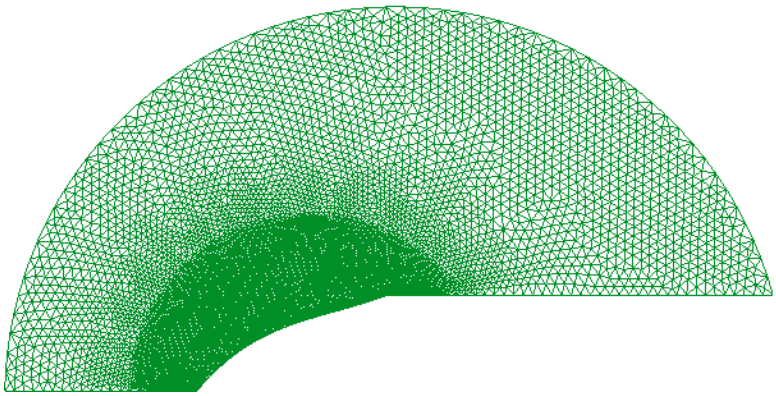


Figure 4.6: Final Refined Mesh

The solution to the unconstrained problem is shown in Figure 4.7. The optimisation methodology has worked well in optimising the shape from a completely random initial starting shape function. Figure A.1 in Appendix A shows the progression of the solution as the mesh was refined. The results here are good, as the initial mesh was able to approximate the optimal shape well after only 10 generations.

4.3.2 Constrained problem

In this problem, we consider that the area beneath the shape function is fixed. Adding a constraint increases the complexity of the problem. Now the objective functional is formulated as,

$$F[\mathbf{y}(\mathbf{x};\underline{\alpha})] = D + \rho_A \|\epsilon_A[\mathbf{y}(\mathbf{x};\underline{\alpha})]\|^2 \tag{4.2}$$

with

$$\epsilon_A = A - A_{goal}$$

where ρ_A is the cross sectional area error term ratio, and ϵ_A is the error in the area (difference between the actual area and the desired area).

The area goal underneath the shape function, A_{goal}, was fixed to 0.75. For a shape closer to an aerofoil, a larger value for the area (between 1 and 2) would simply be used but for the sake of interest a concave shape is investigated. Again the solution was gained in three steps using the same meshes. Here though, the computational time is predicted

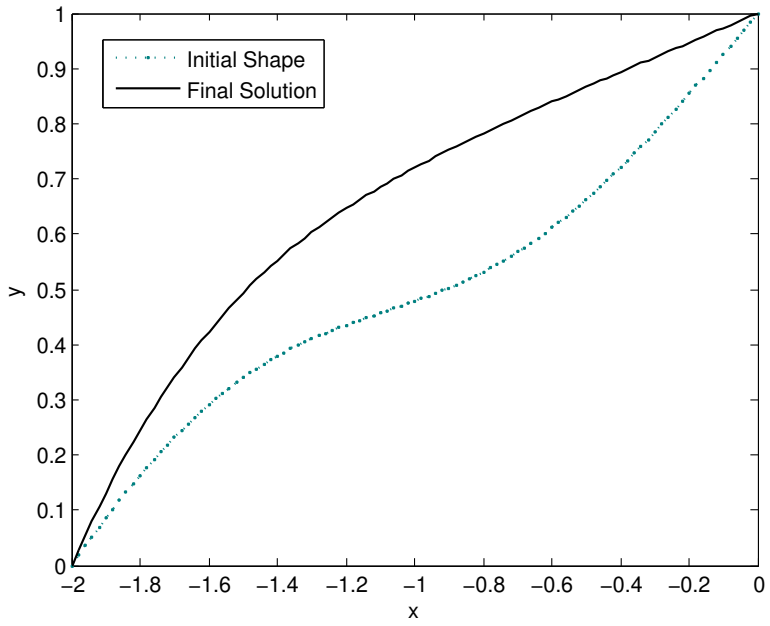


Figure 4.7: Solution for the Unconstrained Problem

to be long due to the added constraint. Therefore 30 Generations of the evolutionary algorithm were used with the initial run, followed by 10 Generations on both of the following runs.

As discussed in Section 2.3.2, the penalty term ratio, ρ_A, is a parameter used to control the penalty terms. The value for the penalty term ratio is determined through a process of trial and error. The area error, ϵ_A, is included in the output at each generation to see how accurately the constraint is adhered to. A value for ρ_A is assigned and the optimisation process is run. The area under each individual solution is approximated using the composite trapezoid rule with 1001 intervals. If ϵ_A in the final solution, is too large then the this error term needs to be more heavily weighted so that there is a larger penalty for the difference between the actual area and the target area. An error of the order 0.001 was viewed as an acceptable error.

When $\rho_A \|\epsilon_A[\mathbf{y}(\mathbf{x};\underline{\alpha})]\|$ is sufficiently small, there is a negligible value added by the penalty term and the drag, D, can then be minimized. If ρ_A is too large however, the drag term will not contribute to the objective functional and the solution will take an impractical amount of time to obtain. Several iterations were run using different values for ρ_A. When ϵ_A was clearly too high or low, then the process was stopped and restarted

with another value for ρ_A. Table 4.3.2 shows the values for ρ_A that were used in the three runs and the associated error, ϵ_A, in the cross sectional area.

The final solution to the constrained problem is shown in Figure 4.8.

Run	**Mesh**	$\rho_\mathbf{A}$	$\epsilon_\mathbf{A}$
1	Coarse Mesh	2	0.0277
2	Intermediate Mesh	5	0.0080
3	Refined Mesh	10	0.0001

Table 4.2: Tuning of the area ratio term

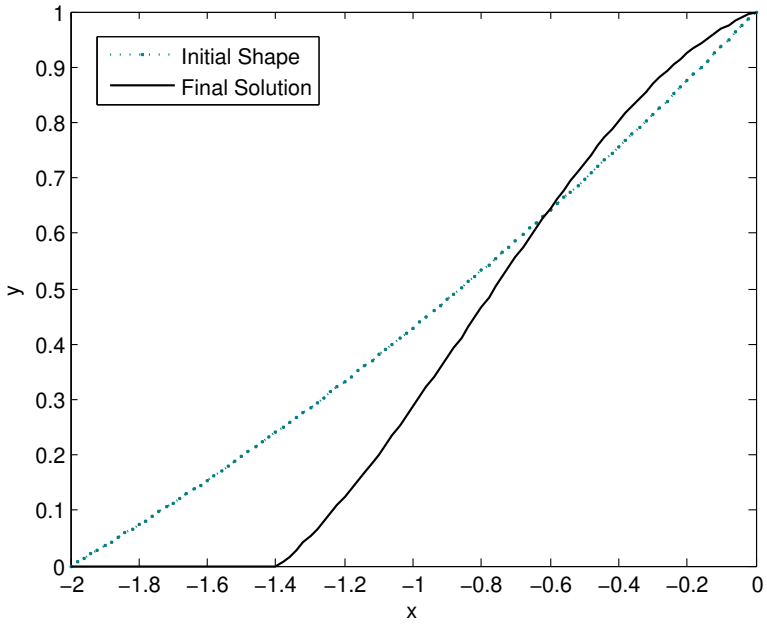


Figure 4.8: Solution for the Constrained Problem

The solution for each mesh in the constrained problem is found in Figure A.2. It can be seen that the solution found from the initial mesh has a high error in the area, given numerically in Table 4.3.2. This indicates that the penalty term for error in the area was not large enough. When ρ_A was increased for the 2nd run using the intermediate mesh, the solution converged to the optimum solution well, with only a minor alteration in the shape after the third run.

Chapter 5

Conclusions and Future Work

Shape optimisation with the use of neural networks has been proven as a potential shape optimisation method for aeronautical applications. The integration of the PUMI Euler solver and the optimisation library into Flood has created a more extensive class library for shape optimisation procedures. In addition, it forms a basis for a black-box optimising environment able to use different solvers and techniques depending on the application. The performance of the evolutionary algorithm was shown to be more reliable in the shape optimisation process than the quasi-Newton method, due to its ability to cope well with noise present in the solution.

There is a lot of scope for further development of the optimisation system. The next step would be to use a more complex neural network to represent the entire aerofoil shape. Also, the optimisation process of the aerofoil would be even more demanding as it would require adding further constraints to the problem such as maximising the Lift-to-Drag ratio (minimising D/L). Due to the large number of computational evaluations needed, this would only be possible using cluster computing facilities to perform parallel evaluations.

However, the technology developed and implemented in this neural-network-based design optimisation procedure offers a unique capability that can be used in other aerospace applications such as external aerodynamics and multidisciplinary optimisation, and has potential applications beyond aerospace design.

Further Work

Currently, the procedure is only suitable for the hypersonic case. For the optimisation of a shape in subsonic flow, the control volume must be much larger. This is because

the pressure will travel faster than the flow, so will propagate forwards as well as back and will be acting in a much larger region of the control volume. Furthermore, the optimisation methodology can eventually be extended to include full viscous effects in the flow using a 3D Navier-Stokes solver available to CIMNE. With all these developments, however, it is clear that the need for the cluster computing facilities is essential if the computational time is to be kept within reasonable bounds.

Bibliography

[1] Proposal for AIRBUS Spain written by CIMNE, CIRA and Univ. of Dortmund, *Development of an advanced Hybrid Evolutionary Algorithm for the optimisation of Aeronautical Configurations in a Multi-objective space (DRAGON)*, 2006

[2] Lopez R. *Flood: An Open Source Neural Networks C++ Library.* www.cimne.com/flood, 2006.

[3] Thamotheram C.P., *Aerodynamic Shape Optimisation using Probabilistic-Stochastic Search Methodologies*, 2002

[4] Fung. A, *Evolutionary Methods for Optimal Shape Design*, 2005

[5] Lopez R., Oñate E., A variational formulation for the multilayer perceptron. *Proceedings of the 16th International Conference on Artificial Neural Networks ICANN 2006*, 2006.

[6] Lopez R., Balsa-Canto E., Oñate E., *Neural networks for variational problems in engineering*, CIMNE, 2006

[7] Lopez R., *Flood A1 - An Open Source Neural Networks C++ Library for Modeling Data*, CIMNE, 2007

[8] Haykin S. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1994.

[9] Šíma J, Orponen P. General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation* 2003; **15**:2727-2778.

[10] Singiresu S. Rao, *Engineering Optimization Theory and Practice*, Third Edition, John Wiley and Sons

[11] Quasi-Newton method, From Wikipedia, the free encyclopedia *http* : $//en.wikipedia.org/wiki/Quasi-Newton_methods$

[12] Pohlheim H., GEATbx - The Genetic and Evolutionary Algorithm Toolbox for Matlab, www.geatbx.com, 2007

[13] Ortega E., Flores R., Oñate E., *An Edge-Based Solver for Compressible Flow*, 2006.

[14] Hirsch C., *Numerical Computation of Internal and External Flows*, Volume 2, John Wiley and Sons Ltd., 1990.

[15] Manzini M., Numerical methods for 1D compressible flows, an interactive book, 2007 $www.crs4.it/HTML/int_book/NumericalMethods/int_book.html$

[16] Donea J., Huerta A., *Neural Networks: A Comprehensive Foundation.*, John Wiley and Sons, 2003, p.159-184.

[17] Mattheij R., Rienstra S., ten Thije Boonkkamp J., *Partial Differential Equations: Modeling, Analysis, Computation*, Society for Industrial and Applied Mathematics, 2005, p.417-439.

[18] Lhner R., *Applied CFD Techniques*, John Wiley and Sons Ltd., 2001

[19] Brown SL, Hull DG. Axisymmetric bodies of minimum drag in hypersonic flow. *Journal of Optimization Theory and Applications* 1969; **3**(1):52-71.

Appendices

Appendix A

Additional Figures

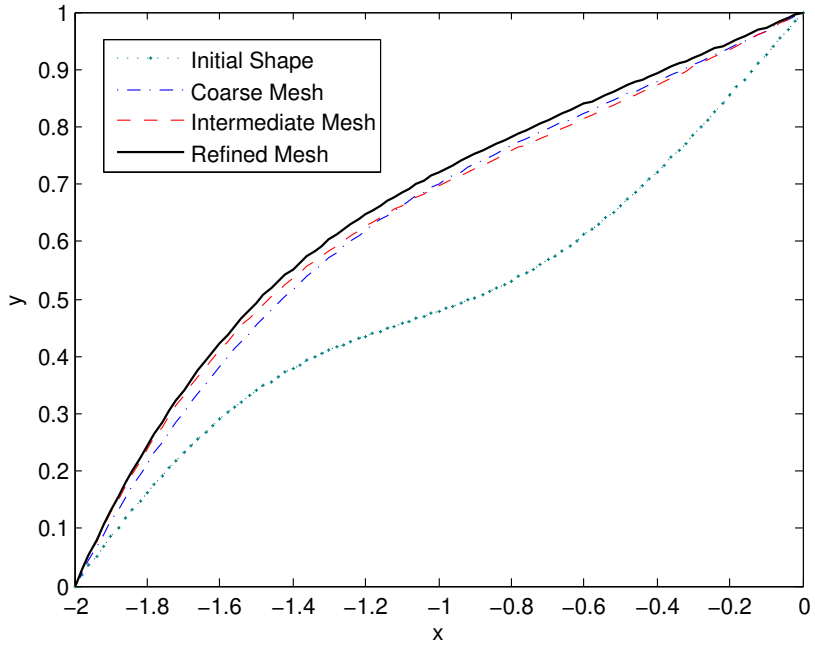


Figure A.1: Shape Functions for the Unconstrained Problem

The solution obtained by each of the meshes in the unconstrained problem can be seen in Figure A.1. The results are encouraging as it is clear that even with the coarse mesh the shape quickly converged from the initial shape to a result close to the optimum. For

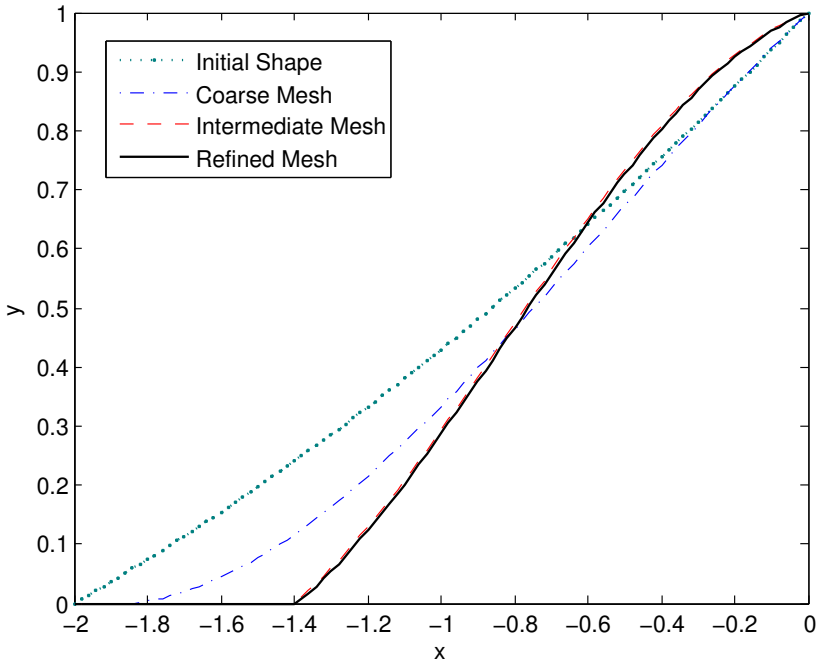


Figure A.2: Shape Functions for the Constrained Problem

the constrained problem shown in Figure A.2, the initial solution has not performed as well and there is high error visible in the cross sectional area. Through tuning of the parameter, ρ_A, the optimum shape is found with the correct area.

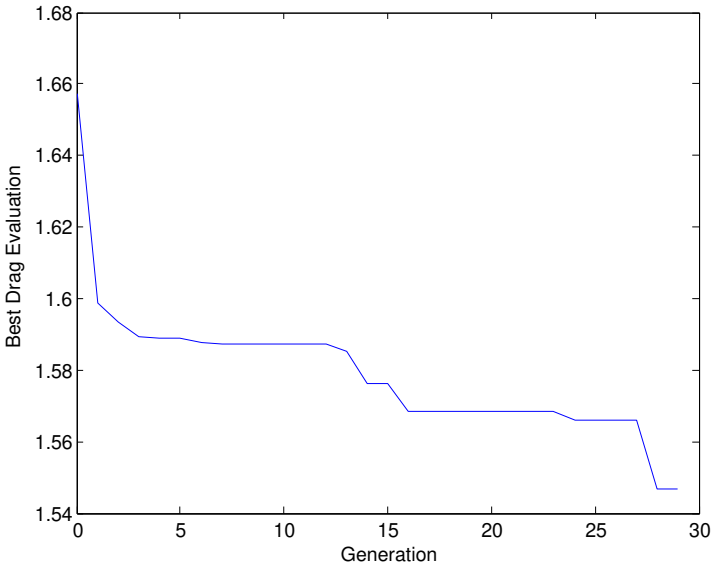


Figure A.3: Best Drag Evaluation for Constrained Problem in 2nd Run: An illustration of the evolutionary training process over 30 generations

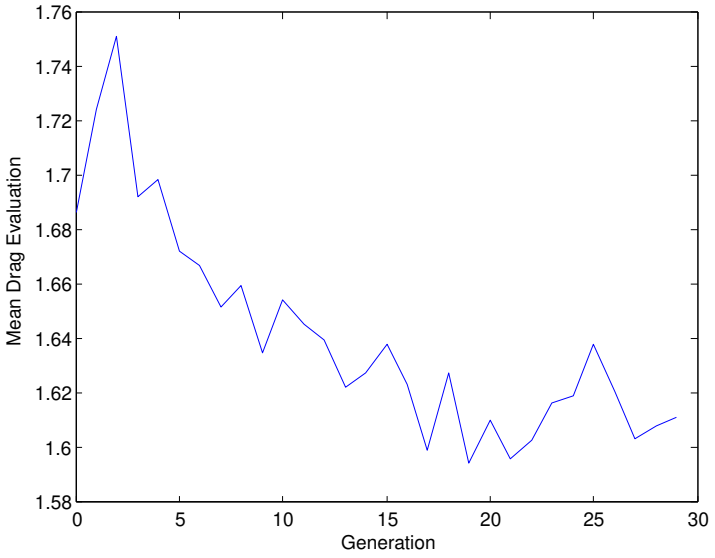


Figure A.4: Mean Drag Evaluation for Constrained Problem in 2nd Run: Illustrating the decreasing mean of the drag evaluation for each generation as the shape is optimised

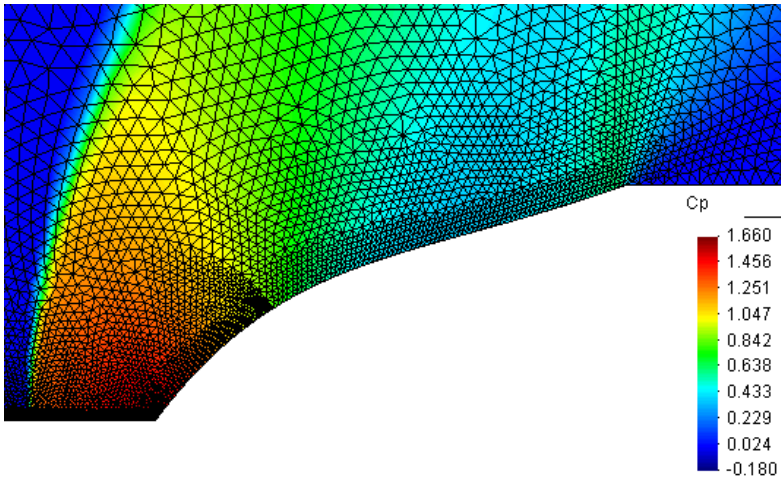


Figure A.5: C_p around the Unconstrained Shape Function with the Intermediate Mesh refined around the area of discontinuity to improve the objective functional evaluation

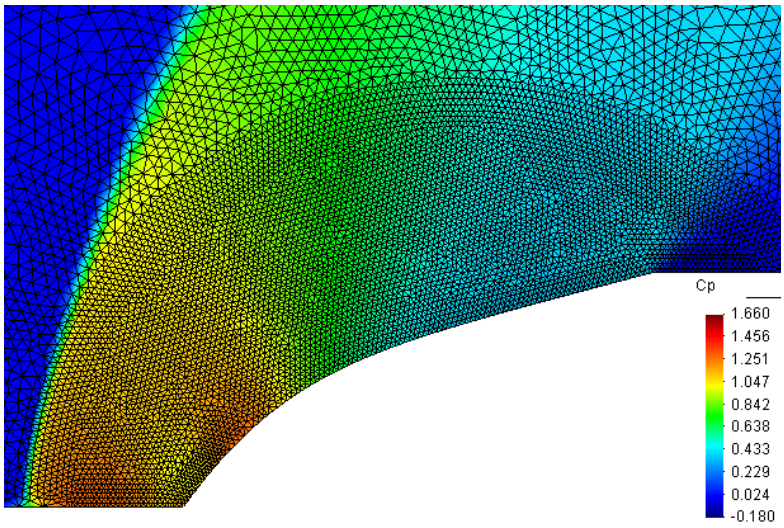


Figure A.6: C_p around the Unconstrained Shape Function with the Refined Mesh

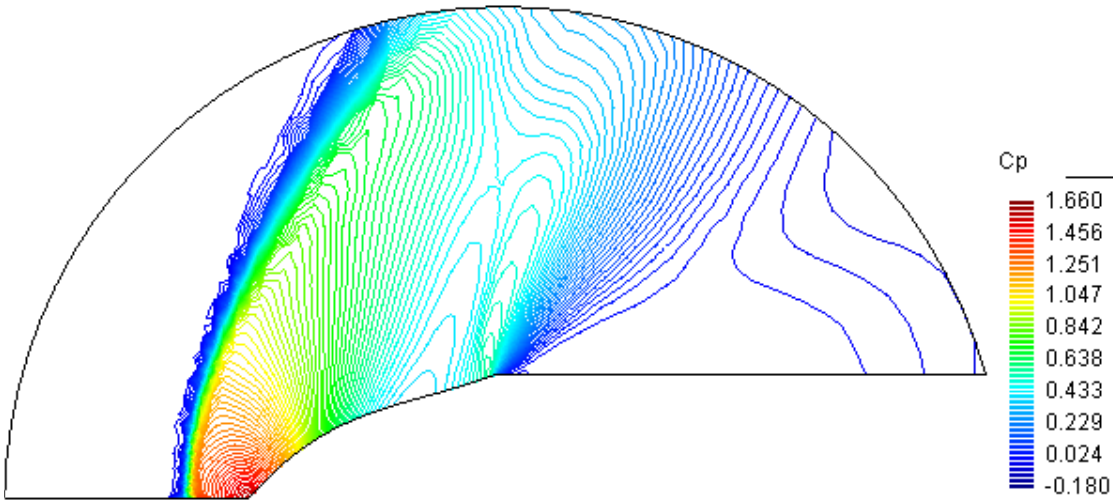


Figure A.7: C_p Contour Lines for Unconstrained Problem using Initial Mesh: Shock is clearly visible with a high pressure region at the front of the shape

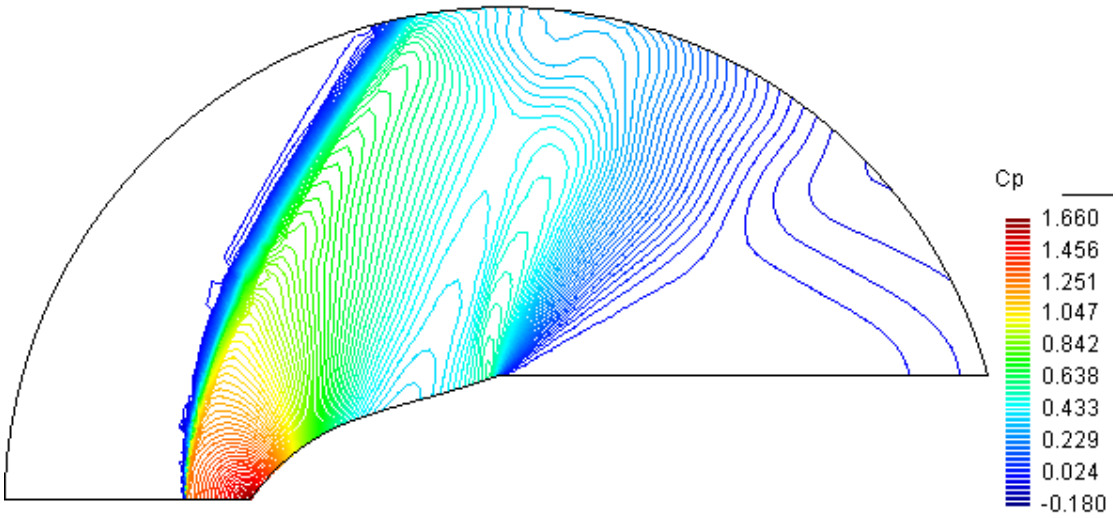


Figure A.8: C_p Contour Lines for Unconstrained Problem using Intermediate Mesh

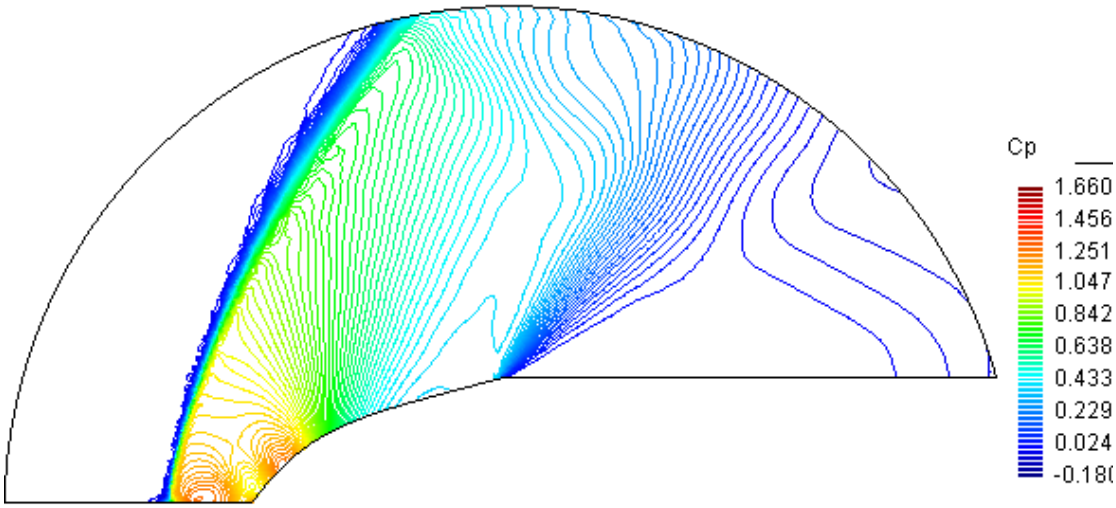


Figure A.9: C_p Contour Lines for Unconstrained Problem using Refined Mesh: Here the C_p is clearly lower at the front of the shape when compared to the previous C_p results

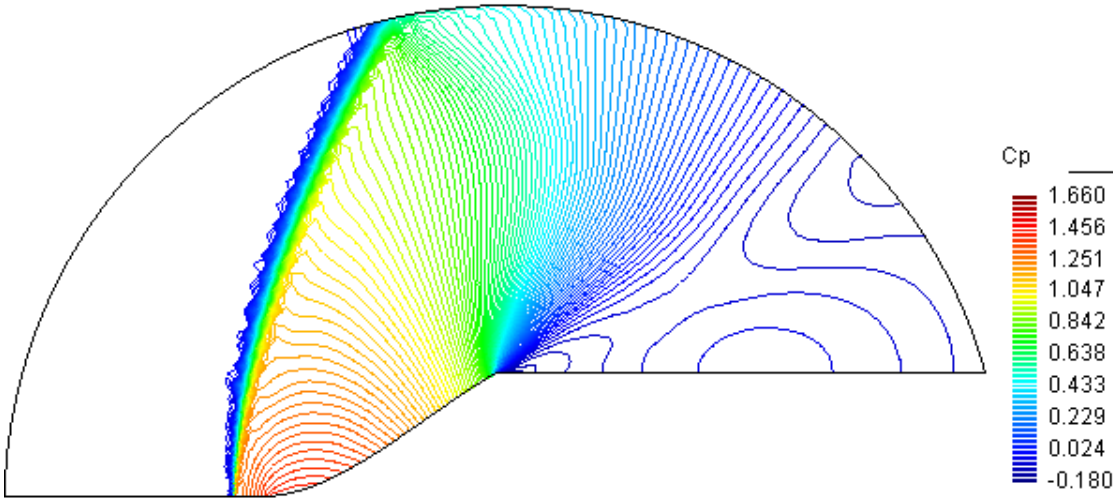


Figure A.10: C_p Contour Lines for Constrained Problem using Initial Mesh

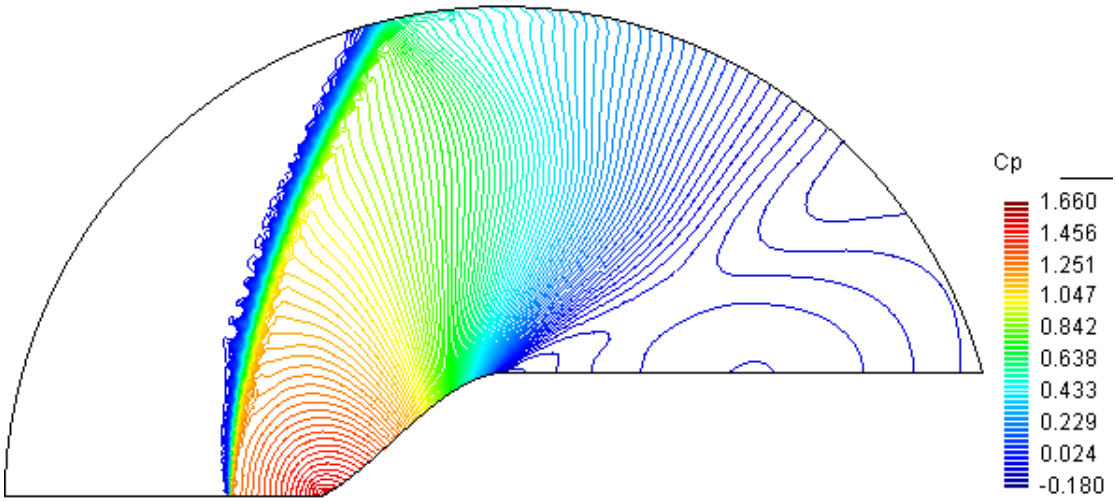


Figure A.11: C_p Contour Lines for Constrained Problem using Intermediate Mesh

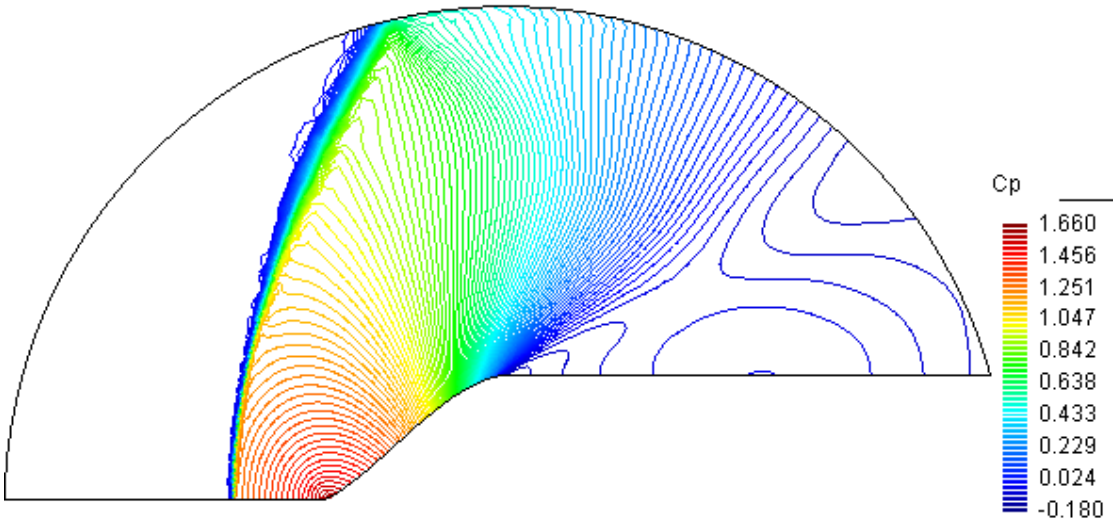


Figure A.12: C_p Contour Lines for Constrained Problem using Refined Mesh

Appendix B

Software

GiD

GiD is an adaptive and user-friendly graphical user interface for pre and post model processing. It can be used for creating model geometry, setting analysis parameters, and allows results to be viewed visually for all types of numerical simulation programs.

GiD can be used for solid and structural mechanics problems, fluid dynamics, electromagnetics, heat transfer, geomechanics, etc. and uses finite element, finite volume, boundary element, finite difference or point based (meshless) numerical procedures.

http://www.gidhome.com.

Flood: An Open Source Neural Networks C++ Library

Flood is a comprehensive implementation of the multilayer perceptron neural network in the C++ programming language. It includes several objective functionals and training algorithms, as well as different utilities for the solution of a wide range of problems. The package comes with extensive documentation [7]. There are two versions of Flood:

- Flood A, for data modeling problems.
- Flood Z, for general variational problems.

The latter is used for the work carried out in this project. For more information visit the http://www.cimne.com/flood/default.asp website.

Appendix C

GiD Batch Files